

A Comparison Framework and Assessment of two Specification Approaches based on UML Scenarios¹

Mohammed Elkoutbi* and Rudolf K. Keller**

* École Nationale Supérieure d'Informatique et d'Analyse des Systèmes
B.P. 713, Agdal, Rabat, Morocco
elkoutbi@ensias.ma

** Zühlke Engineering AG, Zürich-Schlieren, Switzerland, ruk@zuehlke.com, and
Université de Montréal, Dépt. d'informatique, Canada, www.iro.umontreal.ca/~keller

Abstract: In this paper, we present a comparison framework of two proposed approaches for modeling interactive systems using scenarios as defined by the Unified Modeling Language (UML). Scenarios are first transformed into partial specifications and merged to obtain a global specification capturing the behavior of an object of the system or the behavior of the entire system. From the global specification, a User Interface (UI) prototype is then generated and embedded in a user interface builder environment for further refinement.

Keywords: Unified Modeling Language, model transformation, scenario specification, formal methods, user interface prototyping techniques.

1. Introduction

Scenarios have been identified as an effective means for understanding requirements [12] and for analyzing human computer interaction [19]. A typical process for requirements engineering based on scenarios [14] has two main tasks. The first task consists of generating from scenarios specifications that describe system behavior. The second task concerns scenario validation with end users by simulation and prototyping. These tasks remain tedious activities as long as automated tools do not support them.

In interactive systems, the user generally launches many tasks concurrently. He or she interacts alternatively with the UI of these tasks. So we need a formal technique that specifies well this conceptual concurrency. In our work, we adopted Statecharts and Petri nets as the formal techniques for modeling system behavior. Indeed these formalisms are largely used and proved reliable in the field of software engineering. They model well concurrency, they provide structuring possibilities for reducing the size of specifications, and they are supported by both academic and commercial tools.

This paper compares two approaches for requirements engineering which are both based on the UML, the de facto standard notation for object-oriented analysis and design. The first approach, SUIP-PN² (Scenario-based User interface Prototyping version Petri nets), takes a system view and yields a specification of the entire system in form of Petri nets. In contrast, the second approach, SUIP-SC³ (Scenario-based User interface Prototyping version StateCharts), takes an object view and provides for each object of the system a global specification in form of Statecharts. The two approaches provide a four activities process with limited manual intervention for deriving a UI prototype from scenarios and generating a formal specification of the system. The major contribution of this paper lies in the comparative framework of the two modeling approaches based on UML scenarios.

Section 2 of this paper provides an overview of the four activities process leading from scenarios to executable UI prototypes. In Section 3, we compare the two approaches SUIP-PN and SUIP-SC regarding to their view and the formal techniques used. In Section 4, we present some related work, and Section 5 concludes this paper and provides an outlook into future work.

2 Overview of Requirements Engineering Process

We aim to provide an overall process that combines two iterative processes: the formal specification process (see top of Figure 1) and the UI prototyping process (see bottom of Figure 1).

In the step *Scenario Acquisition*, the use case diagram (UsecaseD) and interaction diagrams (SequenceDs or CollIDs) are elaborated to capture use cases and their corresponding scenarios. These diagrams are transformed into formal specifications in the step of *System Specification*. In the step of *Prototype Generation*, a UI prototype is generated from the system specification. This prototype is evaluated and refined by the end user in the step of *Prototype Evaluation*. The step *Verification of Specification* concerns the use of tools and algorithms to edit, check, simulate, and verify the

¹ This research was supported in part by FCAR (Fonds pour la formation des chercheurs et l'aide à la recherche au Québec) and by the SPOOL project organized by CSER (Consortium Software Engineering Research) which is funded by Bell Canada, NSERC (Natural Sciences and Research Council of Canada), and NRC (National Research Council Canada).

² A detailed description of the SUIP-PN approach can be found in [8].

³ A detailed description of the SUIP-SC approach can be found in [9].

properties of the specification. This process is refined dependably of the approach used. Figure 2 gives a detailed view of the process of Figure 1 related to both SUIP-PN and SUIP-SC approaches. The main differences reside in the diagrams and formalisms used to describe scenarios (SequenceDs and CPNs in SUIP-PN, and CollDs and Statecharts in SUIP-SC).

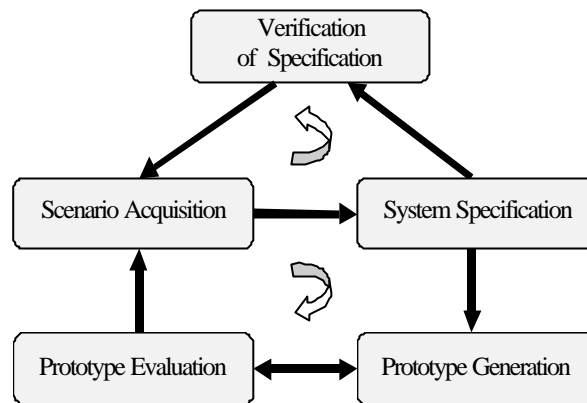


Figure 1: Overview of the proposed process.

2.1 Activities of the SUIP-PN Approach

In the *Scenario Acquisition* activity (Figure 2), the analyst elaborates the UsecaseD, and for each use case, he or she elaborates several SequenceDs corresponding to the scenarios of the use case at hand. The *Specification Building* activity consists of deriving Coloured Petri Nets (CPN)⁴ from the acquired UsecaseD and SequenceDs. CPNs corresponding to the same use case are iteratively merged to obtain an integrated CPN of the use case. Integrated CPNs serve as input to both the *CPN Verification* and the *UI Prototype Generation* activities. During *Prototype Evaluation*, the generated prototype is executed and evaluated by the end user.

The color aspect of CPNs is used to keep track of input scenarios in the resulting specification. A distinct color is allotted to each scenario of the system to retrieve a scenario in the integrated specification. The color tokens are also used to solve the interleaving problem between scenarios. The detailed description of how colors and tokens are used in SUIP-PN approach can be found in [8].

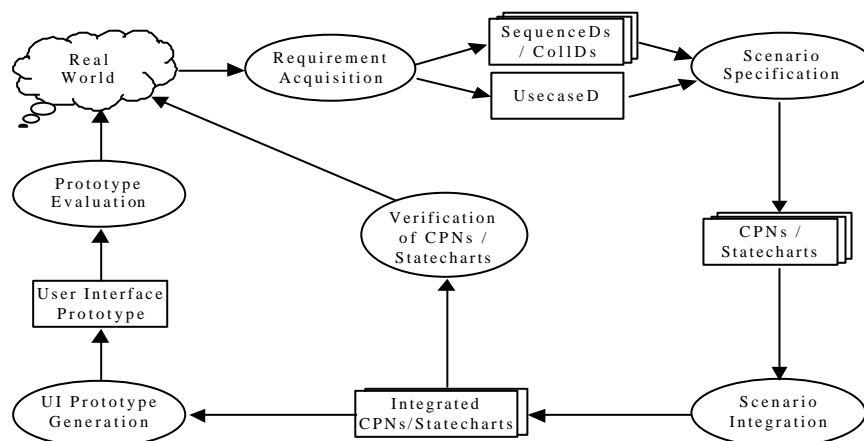


Figure 2: Overview of the SUIP-PN/SUIP-SC approaches.

2.2 Activities of the SUIP-SC Approach

The activities of SUIP-SC are similar to those of SUIP-PN except for the activity of specification building where object specifications are derived from CollDs modeling scenarios into StateCharts. Figure 2 presents the sequence of activities involved in SUIP-SC. In the *Scenario Acquisition* activity, the analyst elaborates the UsecaseD, and for each use case, he or she elaborates several CollDs corresponding to the scenarios of the use case at hand. The *Specification Building* activity consists of deriving and labeling partial StateCharts specifying objects behavior. The partial StateDs

⁴ We use the Jensen formulation of coloured Petri nets and its associated tool designCPN [8].

generated are unlabeled (i.e., their states do not carry names). Given an unlabeled StateD, its states names are identified via the pre-conditions of outgoing and the post-conditions of incoming events. Note that the events in a StateD correspond to the methods of its underlying class found in the ClassD.

In the activity of *scenario integration*, labeled StateCharts of the same object are then iteratively merged to obtain an integrated StateChart modeling the object behavior. Integrated StateCharts serve as input for both the activities of *Verification* and *Prototyping*. During the *Prototype Evaluation*, the generated prototype is executed and evaluated by end users. The detailed description of the entire process of SUIP-SC can be found in [9].

3 Discussion of Approaches

In this section, we compare the two modeling approaches SUIP-PN and SUIP-SC by highlighting their advantages and weakness. We start discussing the UML diagrams (Section 3.1), which we used in our work. Then, SUIP-PN and SUIP-SC will be compared according to their views (Section 3.2) and the specification formalisms used (Section 3.3). We end this section by the discussion of aspects of validation and experimentation (Section 3.4).

3.1 UML Diagrams

Through this work, we used the UML diagrams which relate to the first phases of development: ClassD, UsecaseD, SequenceD, CollD and the StateD.

The UsecaseD captures the services offered by the system, interactions between services, and interactions with the external environment of the system. Interactions between system services are modeled by the relations *uses* and *extends*. These relations give only one simplified sight of the interaction that can really exist between the services of a system. For example, in the case of the automatic teller machine (ATM), the use case *Identification* is used by three other use cases: *Withdrawal*, *Deposit* and *Balance*. Within the relation *uses*, the interaction between the four services of the system is not precisely defined. Really after executing the use case *Identification*, the customer can repeatedly carry out one of other use cases. This can be expressed by using interaction operators like sequentiality (;), choice (|), iteration (*), and concurrence (||) as: [*Identification*; (*Withdrawal* | *Deposit* | *Balance*)*]. So, an extension of the UsecaseD is necessary to capture all these possible interactions between use cases of the same system. This extension can be made by adding some interaction operators to the UsecaseD, or by the definition of a new diagram (use case interaction diagram) similar to CollD where objects are replaced by use cases.

SequenceDs and CollDs were respectively used in SUIP-PN and SUIP-SC. Even if the two diagrams are semantically equivalent, each of them describes better some aspects of scenarios. In SequenceDs, the structure of interactions is specified in a graphical way, which makes sometimes the diagram unreadable. It is even difficult to model interaction as simple as interruptions of iterative messages.

3.2 System and Object Views

In the system view, we were interested to describe the behavior of the overall system. In the vision object we were rather interested to describe the behavior of individually objects. According to our experiences, we present below representative modeling situations where one of the two views is more accurate.

3.2.1 Message Sequencing within Scenarios

In the system view, specifications capture the whole exchanged messages; While, the object view considers only the messages entering and leaving the object at hand. So, there is no guaranty that the global sequencing of messages stays respected. The messages sequencing remains respected locally by the object specification but it may not be respected over all objects specifications.

The object view can cause undesirable behaviors during the simulation. According to the semantics of Statecharts, sending a message means its diffusion to all objects of the system. Normally, this message is sent to a specific object, but it can be interpreted by several objects sensitive to this message.

Within the object view certain behaviors can be declared as non-deterministic during the integration of partial object specifications. During the derivation of an object specification from a scenario, we do not take care of the object sending the message to the studied object. In some situations, one object can receive the same message from two different objects and reacts differently depending on the source of message sent. The integration algorithm will detect these situations as non-deterministic behavior whereas it is not. The SUIP-PN approach is free of all discussed problems in this section.

3.2.2 Labeling States

In the object view, specifications are first derived with no state labels, then they are labeled based on the pre- and post-conditions of messages in the ClassD. This operation is done automatically using an exponential algorithm [7].

In the system view, labels of the derived places of CPNs are deduced from the information put by the analyst in the state tables associated to scenarios. It is possible that the analyst puts in the same table (at different lines) the same information after exchanging two different messages. This can cause the capture of additional behaviors because of the new created cycle in the CPN structure.

3.2.3 UI Prototype Generation

In the object view, interface objects are explicitly identified by the analyst during the requirements step. An UI prototype will be then generated for each object interface by applying the algorithm described in [9]. In the system view, only one UI prototype will be generated for all the system. The two views give similar results for systems having only one actor interacting with a given use case. In case of distributed or collaborative applications, where several actors take part in the realization of a service (use case), the system view will not give satisfactory results. While the object view gives better results with a good identification of interface objects. For example, the gasoline system is a good example where two actors (the customer and the employee) interact with the same use case gas delivery. If we apply the algorithm of UI prototype generation according to the system view, we will obtain a UI prototype where the customer and the employee interactions are merged in the same frame, which is not the case in the real pump system. In the object view, if the analyst identifies correctly interface objects (Pump and TerminalEmployee), a more suitable UI prototype will be generated for each interface object.

3.2.4 Insignificant Behaviors

The object view generates interesting behaviors for interface objects, but not for all objects. In some scenarios, objects can only be solicited by few messages. The generated behavior of these objects can be insignificant specially when the object has not more than one or two states in the corresponding StateDs. This is the case of the Bank object in the ATM system. The behavior of such objects will be better considered at a system level and not at an object level.

3.2.5 Summary

The following table (Table 1) summarizes the discussion of the system and object views. For each line of the table (criterion), the sign - in the column (view) means that the criterion is not supported by this view. The sign + means that it is well supported, and ++ that it is very well supported.

Criterion	System view	Object view
Message sequencing within scenarios	+	-
Labeling states	+	++
UI prototype Generation	+	++
Insignificant behaviors	+	-

Table 1: Some Characteristics of system and object views.

For interactive distributed or collaborative applications, the object view is more suitable. For systems where only one actor interacts with a given use case, the system view will give better results. We can however benefit from the advantages of both views by adopting at first the system view. Once we have obtained the integrated system specification, we can project it onto the various objects of interest in the system. Then, we can apply the algorithm for UI prototype generation on the resulting object specifications.

3.3 Statechart versus Petri Nets

In this section, we aim to compare the two formalisms of specification used in SUIP-PN and SUIP-SC related to a UI point of view. Evaluation of these formalisms according to some criteria such as competition, hierarchy and pre-emption can be found in the work of Holvoet et al. [13]. Holvoet et al. define a new formalism, called Petricharts, to handle interesting features of both Statechart and Petri nets.

3.3.1 Concurrency

Interactive systems present generally interlaced concurrency, except for some special applications such as collaborative ones. The user can launch several tasks in parallel, but he or she cannot interact at the same time with more than one task. This is a conceptual concurrency as defined by Palanque [20]. PNs are known by their support of pure concurrency; thus all transitions having input places with sufficient number of tokens can be fired concurrently. In Statecharts, concurrency is modeled by orthogonal states (and-state) composed of several threads of control. Each thread of control is itself a Statechart. The transitions between states of different threads are not allowed. The communication between threads can be only made through exchanging messages. A transition outgoing from any state

of any thread of an and-state, involves the termination of all the threads of control of the and-state. These restrictions limit concurrency within Statecharts compared to PNs.

3.3.2 Tokens and Multiple Scenarios

The notion of tokens that distinguishes PNs from Statecharts provides the possibility of concurrent executions of distinct scenarios or several copies of the same scenario. Indeed, tokens are used both to control the dynamics of PNs and to model resources of the system. Modeling mutual exclusion is an example where tokens are used to represent the shared resources of the system. Using Statecharts, one can only simulate the execution of one scenario. With PNs, we can simulate simultaneous execution of several scenarios of various use cases, and even several scenarios of the same use case. The number of tokens put in the starting place of the system represents the number of scenarios that the system can carry out simultaneously. Simultaneous simulation of several scenarios makes validation and verification more secure. Errors due to the mutual interaction of the scenarios can be detected during this simulation.

3.3.3 Modal Behavior

By the analogy with a modal window, a modal behavior inhibits concurrency with the remainder parts of the system. In a windowed system, a modal window allows only interactions inside this window. Other windows must remain insensitive to user interactions. A modal behavior can be attributed to a scenario or to a use case of the system. A modal behavior of a use case implies that all its scenarios have modal behaviors. With PNs, modal behaviors are difficult to model. This supposes the existence of a mechanism to freeze tokens in the rest of the system specification. Such mechanisms were the subject of proposal extensions of PNs to support pre-emption [13]. We could however use guard conditions to model a modal behavior, but that would require the addition of conditions in all transitions of the PN. Historical state of Statecharts makes it easy to model modal behaviors. When a modal behavior is initiated, an event is forwarded to all concurrent threads to enter their historical states. As soon as the modal behavior finished, a second message is forwarded to threads to reactivate them again. So the SUIP-SC approach will be more appropriate to model modal behaviors than SUIP-PN.

3.3.4 Verification of Specifications

The resulting specifications derived from scenarios in SUIP-PN and SUIP-SC can be verified using existing tools.

In SUIP-SC, we defined a verification framework [7] checking specifications along the activity of scenario specification. We check scenario specifications before and after their integration to detect easily scenarios that cause incoherence in the integrated specification. In our actual version of SUIP-PN, we used the designCPN to check and the simulation of the resulting hierarchical CPN.

In SUIP-SC, we developed a new algorithm based on pre- and post-conditions semantic [7]. This algorithm comes to complement existing ones by detecting incoherence and incompleteness among the integrated scenarios. It is based on the semantics given to states in SUIP-SC that are labeled by the pre- or post-conditions of the incoming and out-coming events.

3.3.5 Tool Support

Beyond STATEMATE [11], Statecharts are lightly supported by tools, while for PNs the number of existing tools exceeds the hundred (tools for edition, checking and simulation). Moreover extension works of PNs are numerous: PNs with predicates, coloured PNs, temporal PNs, stochastic PNs and others. Evolution of SUIP-PN to capture new aspects like real time constraints for example would be easier regarding the existing extensions of PNs and the supporting tools.

3.3.6 Summary

Table 2 summarizes main differences Statecharts and PNs according to criteria previously discussed. We can say that PNs are more accurate to use in UI modeling.

Criterion	PN	Statechart
Concurrency	++	+
Tokens and multiple scenarios	+	-
Modal behavior	-	+
Verification of specifications	+	+
Tool Support	++	+

Table 2: PNs and Statecharts from a UI point of view.

3.4 Validation and experiments

The proposed ideas and developed algorithms were experimented in many systems as : the ATM system, the system of library and the system of gasoline station. The execution time of the various developed algorithms was about few seconds, knowing that the number of scenarios varies from five to eight scenarios per system.

It is well known that scalability is an inherent problem when dealing with scenarios for large applications. Our approach eases this problem by integrating scenarios per use case, rather than treating them as an unstructured mass of scenarios.

3.4.1 Approach SUIP-PN

In the SUIP-PN approach, we used a several existing tools (designCPN [5], Sxtool [2], XML4J [15] and Visual Café [23]) interlaced by the application of developed algorithms. The first algorithm reads and integrates scenario specifications of in PNs in a XML format, and produces the integrated specification of all use cases of the system in the same format (XML). The second algorithm takes as input the integrated specifications and produces the UI prototype of the system. The two algorithms are polynomial. Their establishments count approximately 4000 lines of Java code (comments included).

3.4.2 Approach SUIP-SC

In the SUIP-SC approach, a succession of algorithms is automatically applied to scenarios of the system to produce the integrated specifications of all objects, and then to generate UI prototypes from interface objects. This approach brings more mechanization than the SUIP-PN. The SUIP-SC inputs are the ClassD describing data and relationship, and CollIDs describing scenarios of the system. Five algorithms are then applied to obtain the desired results. The first algorithm transforms CollIDs into partial objects specifications. The second algorithm labels the partial specifications produced by the first algorithm. The third algorithm integrates labeled partial specifications of the same object. The fourth algorithm generates starting from the integrated specification of each interface object a visual UI prototype. The fifth algorithm checks incoherencies in the produced specifications. All the above mentioned algorithms are implanted in Java with a total of approximately 4500 lines of code (comments included).

In the two approaches (SUIP-PN and SUIP-SC), the UI prototype is generated in a Java code compatible with the interface builder Visual Café [23] which is only used to improve the visual aspect of the UI prototype by adding titles, comments, colors, etc.

4 Related Work

In this section, we review some related work in the area of scenario integration and automatic UI generation from specifications.

4.1 Scenario Integration

In the area of scenario integration, most research has only addressed the problem of sequential integration [6, 14, 17, 21], and few researchers have been interested in a more general form of integration [4, 10].

Hsia [14] describes scenarios as directed trees, where nodes represent system states and the arcs represent events. The sequential composition of two scenarios means merging their corresponding trees. The merged tree is transformed into a regular grammar from which a state machine will be generated.

Koskimies [17] presents an algorithm for synthesizing an object Statechart specification from a list of scenarios. He tries to infer a Statechart able to execute all traces corresponding to input scenarios using the Bierman's algorithm [Bie75].

Desharnais [6] defines a scenario as the union of two relations R_e and R_s . R_e represents the environment relation which captures all possible actions and R_s represents the relation corresponding to the system reaction.

Somé [21] represents scenarios as timed automata. In his work, he is mostly interested in timing issues. He defines a scenario integration algorithm to generate a timed automaton modeling the behavior of the entire system. The generated specification does not capture hierarchy nor concurrency features.

In [10], Glinz gives a way for composing scenarios in form of Statecharts using some operators (conditional, iterative and concurrent). He does not support scenarios overlapping. His approach is somewhat close to our use case level.

Dano [4] proposed a use case formalization with Petri nets, he defines a list of temporal relations between use cases (begin at the same time, end at the same time, one after the other, etc.). He used conditions and assumptions to link all the possibilities (scenarios) of a use case.

All the discussed approaches address only one level (use case or scenario), and generate a specification that captures more than the initial scenarios because of the interleaving problem between scenarios. In SUIP-PN [8] and SUIP-SC [9] we clearly address the two levels use case and scenarios, and we resolve the interleaving problem.

The following table summarizes characteristics of the discussed approaches related to criteria like sequentiality, concurrence, overlapping, hierarchy, interleaving, vision, and level of detail.

Criterion Approaches	Sequentiality	Concurrence	Overlapping	Hierarchy	Interleaving	Vision	Level
Hsia [14]	+	-	+	-	-	System	SC
Koskimies [17]	+	-	+	-	-	Object	SC
Desharnais [6]	+	-	+	-	-	System	SC
Somé [21]	+	-	+	-	-	System	SC
Glinz [10]	+	+	-	+	-	System	UC
Dano [4]	+	+	+	-	-	System	UC
SUIP-PN [8]	+	+	+	+	+	System	UC & SC
SUIP-SC [9]	+	+	+	+	+	Object	UC & SC

Table 3: Characteristics of integration approaches.
SC: Stands for scenario, UC: Stands for use case.

4.2 Automatic Generation of UIs from Specifications

A number of methods have been suggested to derive UI prototypes from system specifications. Typically, data attributes serve as input for the selection of interaction objects according to some rules based on style guidelines such as *CUA* (Common User Access), *OpenLook* [22], and *Motif* [18]. Such methods include the *Genius*, *Janus*, and *TRIDENT* approaches.

In *Genius* [16], the application domain is captured in data models that are extended entity-relationship models. The analyst defines a number of views, where each view is a subset of entities, relationships, and attributes of the overall data model, and he or she specifies how these views are interconnected by means of a Petri-net based dialogue description. From these view and dialog specifications, *Genius* generates the UI. Note that the specification process is completely manual.

Janus [1] derives different windows of a UI from object models as introduced by Coad and Yourdon [3]. Non-abstract classes are transformed into windows, with attributes and methods marked as irrelevant for the UI being ignored in the transformation process. *Janus* does not address the dynamic aspect of UIs.

Note that, in contrast to our approach, both *Genius* and *Janus* use data structure specifications for UI generation, but ignore task analysis altogether. As a consequence, such methods are little useful for systems other than data-oriented applications.

TRIDENT [2] leverages both task analysis and functional requirements analysis. Task analysis proceeds by decomposing the application into interactive tasks and by determining task attributes such as importance and user stereotype (user's task experience, user's system experience, etc.). Functional requirements analysis builds an entity-relationship model for the data and extracts from task analysis the tasks that should be treated as internal functions. An activity-chaining graph is drawn to connect interactive tasks to data and functions. This graph serves as the input for the selection of different windows, referred to as *presentation units*. *TRIDENT* provides three types of assistance in defining presentation units, but is not specific about this assistance, nor does it address the dynamic aspect of UIs.

8 Conclusion

In addition to the suggested iterative process refined in SUIP-PN and SUIP-SC approaches, the most interesting features of this work lie: in the automation brought upon by the developed algorithms, in the use of UML scenarios addressing not only sequential scenarios but also scenarios that exhibit concurrent behaviors, in the derivation of executable prototypes that are embedded in a UI builder environment for refinement, and in the comparative support of SUIP-PN and SUIP-SC approaches.

We plan to extend this work to capture real time interactions. It will be also interesting to investigate a global approach for scenario management where grained scenario interactions are considered. In our current work scenarios are supposed to be independent from each others. Coherence between formal and semi-formal models, links between specifications and usability of UI prototypes can also be tracks for future work.

Bibliography

- [1] Balzert H., "From OOA to GUIs: The Janus System", *IEEE Software*, 8(9), February 1996, pp. 43-47.
- [2] Clark J., *SX : An SGML System Conforming to International Standard ISO 8879*, <<http://www.jclark.com/sp/sx.htm>>.
- [3] Coad P. and Yourdon E., *Object-oriented Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1991.
- [4] Dano B., Briand H. and Barbier F. : An Approach based on the Concept of Use Case to Produce Dynamic Object-Oriented Specifications, In proceeding of the Third IEEE International Symposium on Requirements Engineering, pp.54-64, Annapolis (1997).

- [5] designCPN: version 3, Meta Software Corp. <<http://www.daimi.aau.dk/~designcpn>>.
- [6] Desharnais J., Khediri R., Frappier M. and Mili A. : Integration of Sequential Scenarios. *Transactions on Software Engineering*, 24(9):695-704, September 1998.
- [7] Elkoutbi M., User Interface Engineering based on Prototyping and Formal Methods. PhD thesis, Université de Montréal, Canada, March 2000. French title: Ingénierie des interfaces usagers à l'aide du prototypage et des méthodes formelles.
- [8] Elkoutbi M. and Keller R. K., User Interface Prototyping based on UML Scenarios and High-level Petri Nets. In *Application and Theory of Petri Nets 2000 (Proc. of 21st Intl. Conf. on ATPN)*, Springer-Verlag LNCS 1825, pp. 166-186, Aarhus, Denmark, June 2000.
- [9] Elkoutbi M., Khriiss I., and Keller R. K., Generating User Interface Prototypes from Scenarios, in *Proceedings of the Fourth IEEE International Symposium on Requirements Engineering (RE'99)*, pages 150-158, Limerick, Ireland, June 1999.
- [10] Glinz M.: An Integrated Formal Model of Scenarios based on Statecharts. In *Fifth European Software Engineering Conference, Lecture Notes in Computer Science, Vol. 989*, pp. 254-271, Springer-Verlag (1995).
- [11] Harel D., Lachover H., Naamad A., Pnueli A., Politi M., Sherman R., Shtull-Trauring A. and Trakhtenbrot M.: STATEMATE: A Working Environment for The Development of Complex Reactive Systems. *Transactions on Software Engineering*, 16(4):403-414, April 1990.
- [12] Heitmeyer C., Kirby J., Labaw B., and Bharadwaj R., "SCR*: A Toolset for Specifying and Analyzing Software Requirements", *Proc. of the 10th Annual Conference on Computer-Aided Verification, (CAV'98)*, pp. 526-531, Vancouver, Canada, 1998.
- [13] Holyoet T. and Verbaeten P., Petri Charts : An Alternative Technique for Hierarchical Net Construction. In *Proceedings of the 1995 IEEE Conference on Systems, Man and Cybernetics (IEEE-SMC'95)*, pp. 22-25, Vancouver, Canada, 1995.
- [14] Hsia P., Samuel J., Gao J., Kung D., Toyoshima Y., and Chen C: Formal Approach to Scenario Analysis. *IEEE Software*, 11(2):33-41, March 1994.
- [15] IBM, XML Parser for Java, in *Java Report's February 1999*, <<http://www.alphaworks.ibm.com/formula/xml>>.
- [16] Janssen C., Weisbecker A., and Ziegler U., "Generating User Interfaces from Data Models and Dialogue Net Specifications", *Proc. of the Conference on Human Factors in Computing Systems (CHI'93)*, pp. 418-423, Amsterdam, The Netherlands, April 1993.
- [17] Koskimies K. and Makinen E.: Automatic Synthesis of State Machines from Trace Diagrams. *Software Practice & Experience*, 24(7): 643-658, 1994.
- [18] Open Software Foundation, *OSF/Motif Style Guide*, Prentice Hall, Englewood Cliffs, NJ, USA, 1990.
- [19] Palanque P. and Bastide R., Modeling clients and servers in the Web using Interactive Cooperative Objects, *Formal Methods in HCI*, pp.175-194, Springer-Verlag, 1997.
- [20] Palanque P. and Bastide R.: Specifications Formelles pour l'Ingenierie des Interfaces Homme-Machine. In *Proceedings of IHM'97*, Poitiers, France, 1997.
- [21] Somé S.S., Dssouli R., and Vaucher J.: Toward an Automation of Requirements Engineering using Scenarios. *Journal of Computing and Information. Special issue: ICCI'96, 8th International Conference of Computing and Information*, 2(1):1110-1132, 1996.
- [22] Sun Microsystems, Inc. and AT&T, *OPEN LOOK GUI Application Style Guidelines*, Addison-Wesley, USA, 1990.
- [23] Symantec, Inc, *Visual Café for Java: User Guide*, Symantec, Inc., 1997.